

L03

Responding to Events

L03- Introduction

Clive Maynard



L03 Responding to Events

All computer applications must respond to external events and these events may be:

- keypad inputs for a burglar alarm, microwave oven, TV remote control, etc.
- data received from a hard disk,
- engine temperature input via an analogue to digital converter
- the list is endless (almost).

This section looks at how a computer can access and respond to this data in a timely fashion. These techniques will be essential for all microcontroller applications that you design.

References:

Marilyn Wolf, Computers as Components: Principles of Embedded Computer System Design (4th Edition), Morgan & Kaufmann

David A. Patterson and John, L. Hennessy, Computer Organization and Design: The hardware/Software Interface(5th Edition), Morgan & Kaufmann



L03 Responding to Events

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act).

The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.



Synchronous Communications

The sender/receiver is considered fast enough to provide/accept the data as fast as it is required/sent or a clock signal is available to pace the transfer.

Either way there is no direct feedback from the receiver to the sender. The transfer of data over a microprocessor bus is usually synchronous (it is assumed that the memory or I/O devices accessed by the microprocessor are fast enough to perform the required data transfers). **However, there are some microprocessors which have employed an asynchronous bus with a handshake signal to synchronise the transfer, the Motorola 68000 is a classic example.**

As we will see, responding to external events usually requires some kind of bi-directional interaction between the computer and the outside world.



An Example of Synchronous Transfer

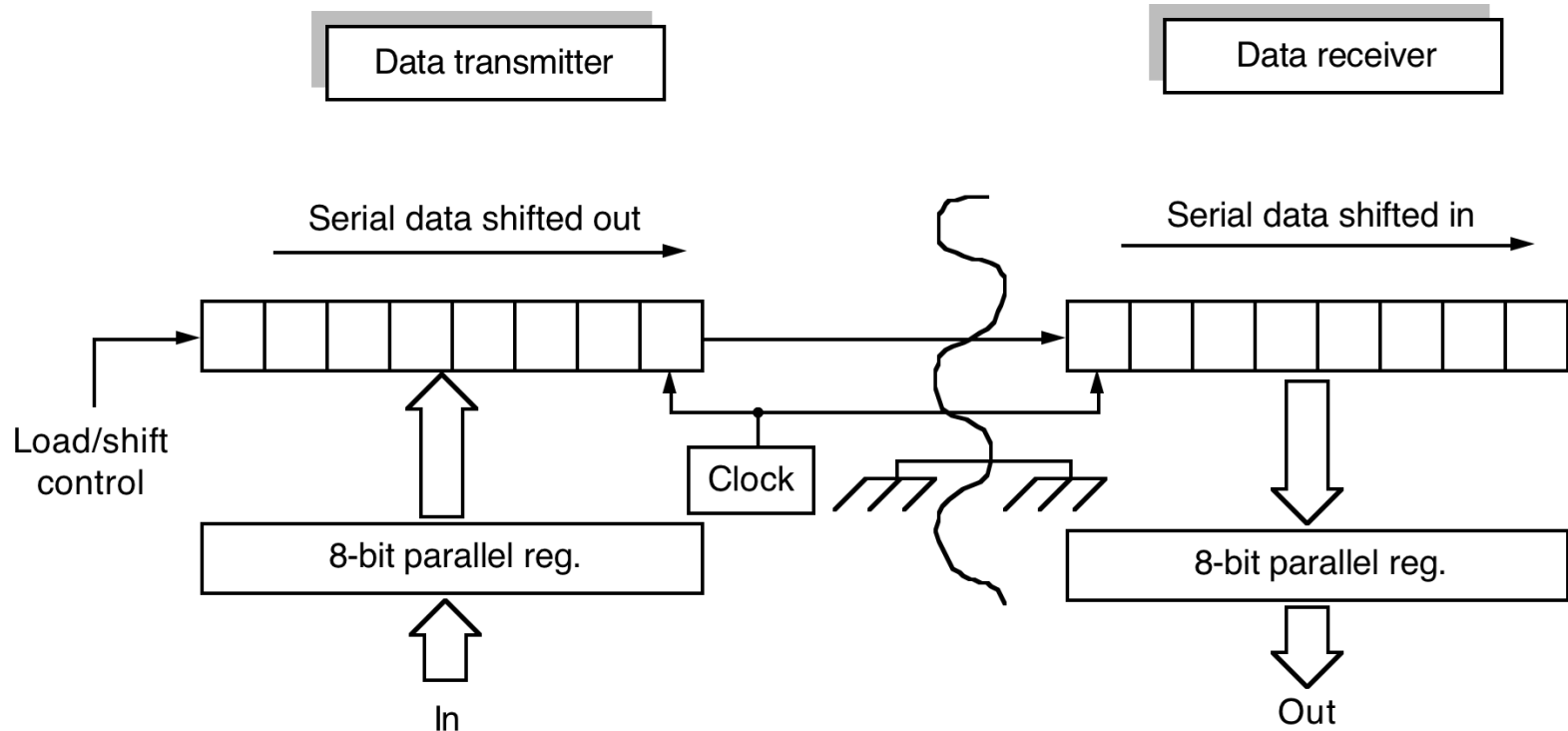
- 1) Parallel data from a computer is buffered in an 8-bit parallel register.
- 2) When the last bit of the previous character leaves the shift register then the new data is parallel loaded into the shift register.
- 3) This data is then shifted out serially at a rate determined by the clock. The clock signal is also supplied to the receiver (requires an extra line between transmitter and receiver) and clocks the data into the receiver shift register.
- 4) When the shift register is full the data is parallel loaded into another 8-bit parallel register where it is read by the computer at the receiving end.

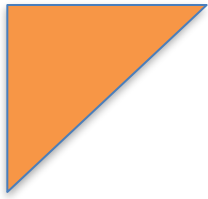


An Example of Synchronous Transfer Continued

- 5) Transmitter and receiver are synchronised by sending sync characters from one end to the other until the receiver recognises them. This takes place when communications is first established or when a transmission error occurs.
- 6) Synchronous serial transmission is not as common as asynchronous transmission because it requires the clock to be transmitted separately.
- 7) If there is no facility for stopping the clock then the transmitter must send null characters until there is data to send.

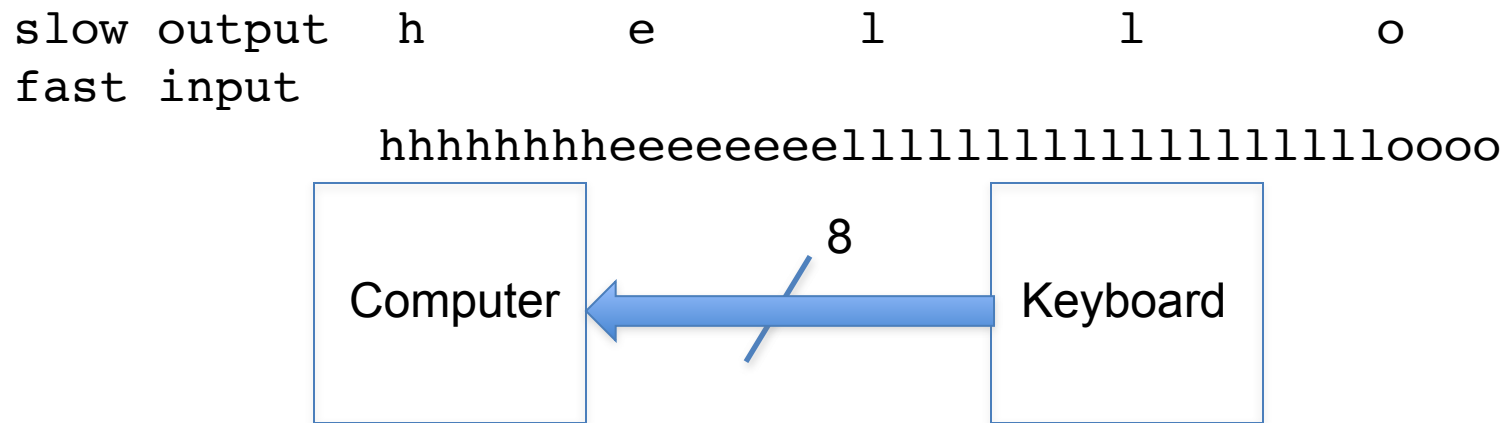
Synchronous Transfer schematic





The need for synchronisation in an asynchronous transfer

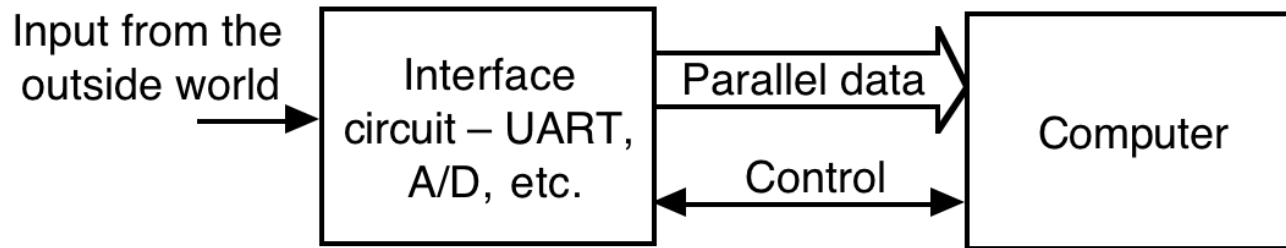
Consider a peripheral device which is sending ASCII characters to the computer as parallel digital data. Because the computer is much quicker than the peripheral device it reads the same character several times.



How does the computer work out where one character finishes and the next starts?

Input to the Computer

Asynchronous handshake

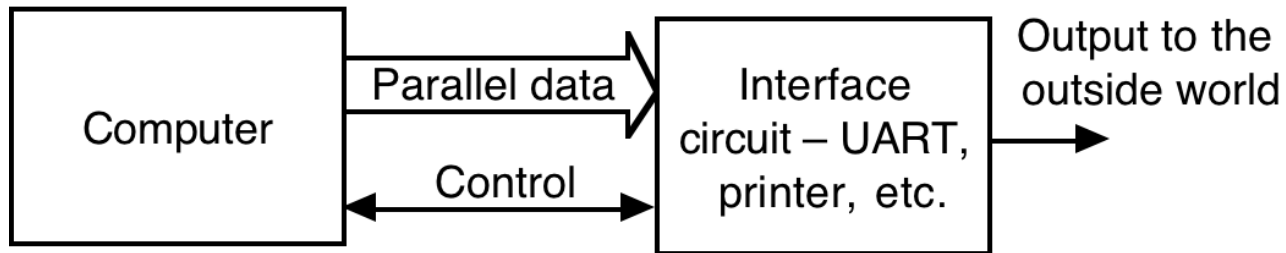


IDLE
DATA READY
DATA TAKEN
IDLE

Interface says 'data not available'
Interface says 'data now available'
Computer says 'data has been taken'
.....

Output From the Computer

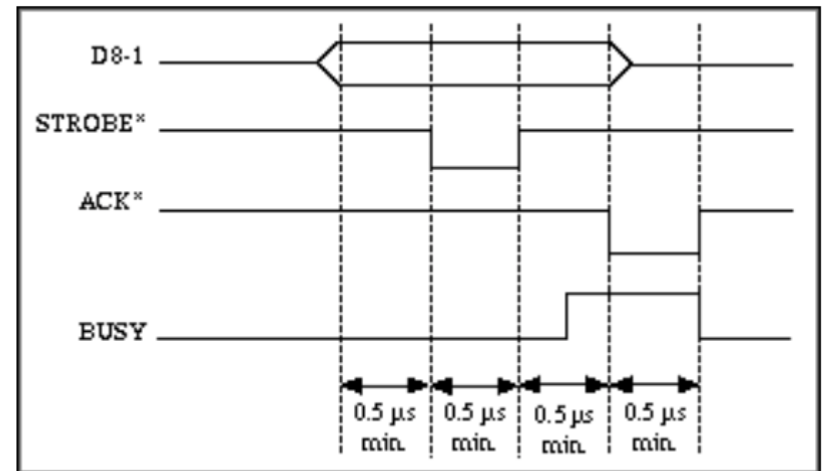
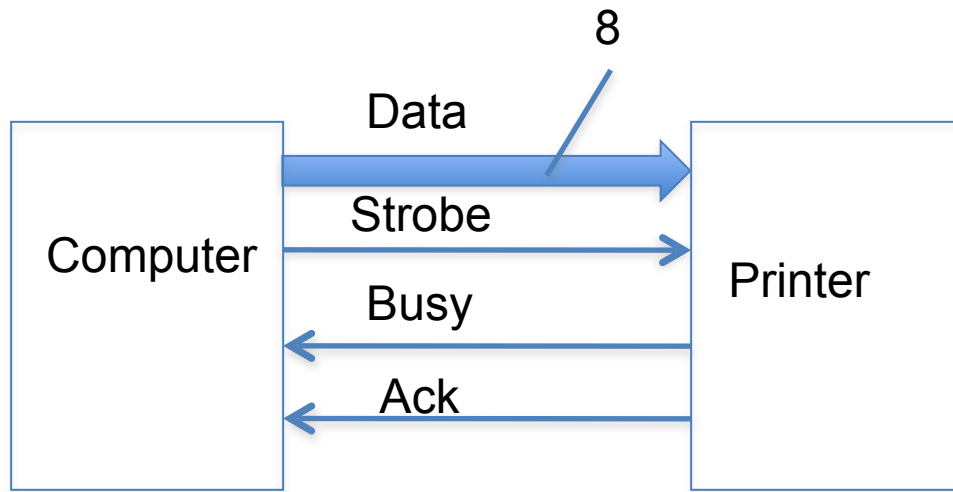
Asynchronous handshake



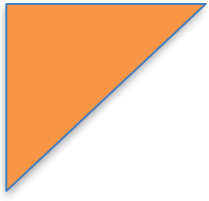
IDLE
DATA READY
DATA TAKEN
IDLE

Interface says 'ready to receive data'
Computer says 'data now available'
Interface says 'data has been taken'
....

Example: Centronics Printer Interface (1970)



Incorporated into the IEEE1284 interface specifications



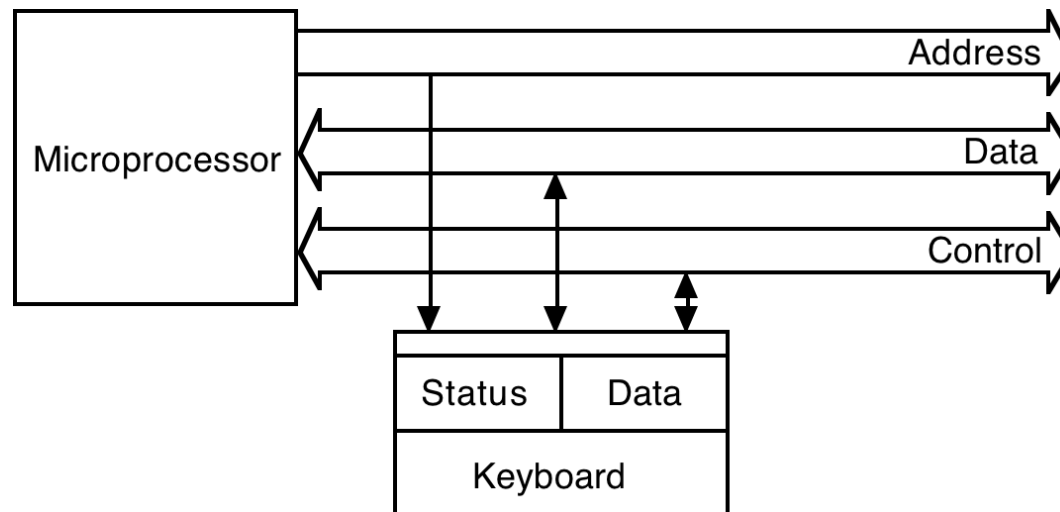
L03-2

Polled I/O

Clive Maynard

Polled I/O

As an example we will consider a keyboard interfaced to a microprocessor bus.



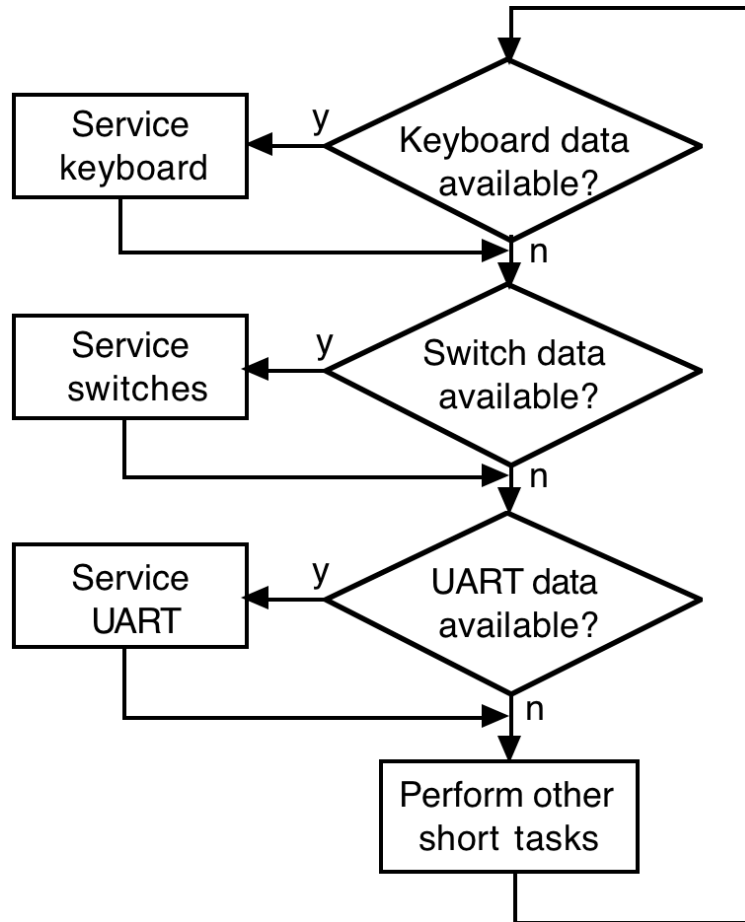


Scenario

- The microprocessor tests the 'key pressed' bit in the keyboard status register.
- If 'key pressed' is asserted then branch to the keyboard service routine.
- If the keyboard is not ready then test the next peripheral system or loop back to the first test.

```
movia      r17, 0x10000050      /* pushbutton KEY base address */
.
.
ldwio      r5, 0(r17)           /* load pushbuttons */
andi       r5, r5, 0b10         /* check button #1*/
bne        r5, r0, SERVICE_BUTTON /* branch to service routine if pressed*/
```

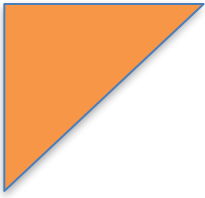
A Polling Loop



The resources required for programmed I/O:

In addition to the polling program, this requires a status register containing a readable 'key pressed' bit.

How could the polling loop be modified to increase the priority (probability of being serviced) for the keyboard?



Polling Latency

When responding to events in the outside world, an important consideration is latency.

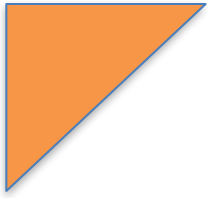
There is no general agreement on the exact definition of latency. The definitions are: (1) Latency is the time between a significant change of input to the microprocessor occurring and the microprocessor responding to it (starting to execute code servicing the event).

(2) An alternative definition includes the time taken to execute the service routine as well.

BE SURE TO SPECIFY WHICH DEFINITION YOU ARE USING

Examples where a computer system should respond sooner rather than later:

- 1) Motorcar airbag deployment
- 2) Mobile phone response to touch gestures on its screen
- 3) Phalanx gun response to detection of an Exocet missile (Historical reference)
- 4) Etc.



Polling Latency Example

Example: if a microprocessor is executing the following polling loop what is the minimum delay and maximum delay between bit zero of the I/O port being set and starting to process the service routine? (Using latency definition 1)

loop:	in	al,[dx]	;dx contains the address of the I/O port	(8 T-states)
	test	al,01h	;test bit zero	(4 T-states)
	jz	loop	;loop back if the bit is not set	(16/4 T-states)

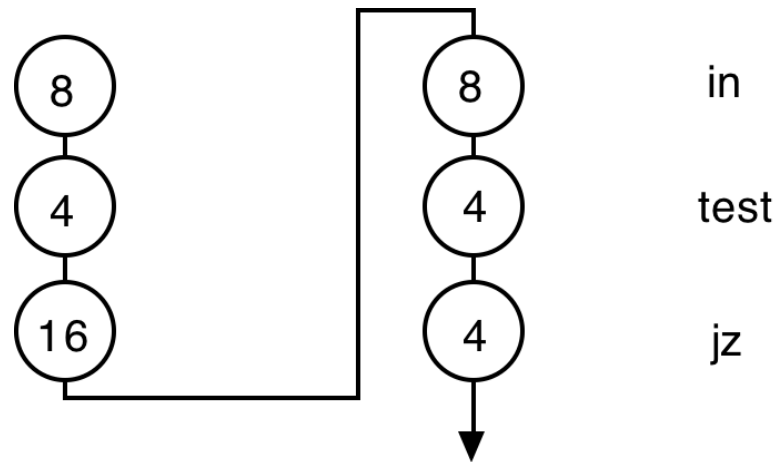
SERVICE ROUTINE

This example is given in Intel 8086 assembler but the meaning of the instructions should be obvious.

T- states are cycles of the microprocessor clock. To work out the actual time the number of T states must be multiplied by the clock period.

Worst Case

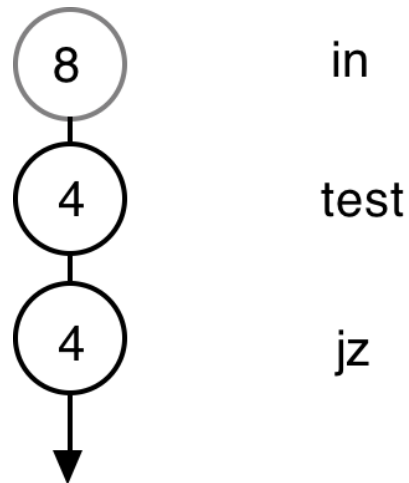
In the worst possible case the 'in' instruction is assumed to sample the I/O port at the start of the instruction and the status bit is set shortly afterwards (ie the microprocessor just misses the change in the I/O port value):



maximum latency = 44T

Best Case

In the best possible case the 'in' instruction samples the I/O port at the very end of the instruction and the status bit is set shortly before this:



Minimum latency = $8T$

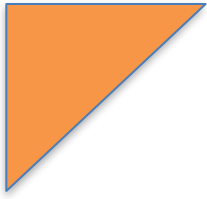


T-States

In this example T-states are cycles of the 8086 clock. Assuming a 5MHz clock (a really old microprocessor) the time required for :

$$8T = 8 / (5 \times 10^6) = 1.6\mu s$$

$$44T = 44 / (5 \times 10^6) = 8.8\mu s$$



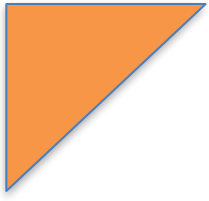
Evaluation of Polled I/O

Advantages

- Can respond very quickly for a small number of I/O devices.
- An arbitrary number of I/O devices can be supported.
- Very flexible - I/O devices can be readily added/deleted to/from polling list and priority adjusted.
- Requires little additional hardware.
- Predictable execution.

Disadvantages

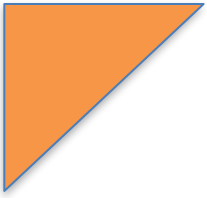
- Consumes computer time even when no I/O operations are occurring.
- Response time is very slow for a large number of I/O devices.
 - Difficult to do other things whilst polling.
- Urgent I/O requests cannot suspend the servicing of low priority tasks.



L03-3

Interrupt I/O

Clive Maynard



Interrupt I/O

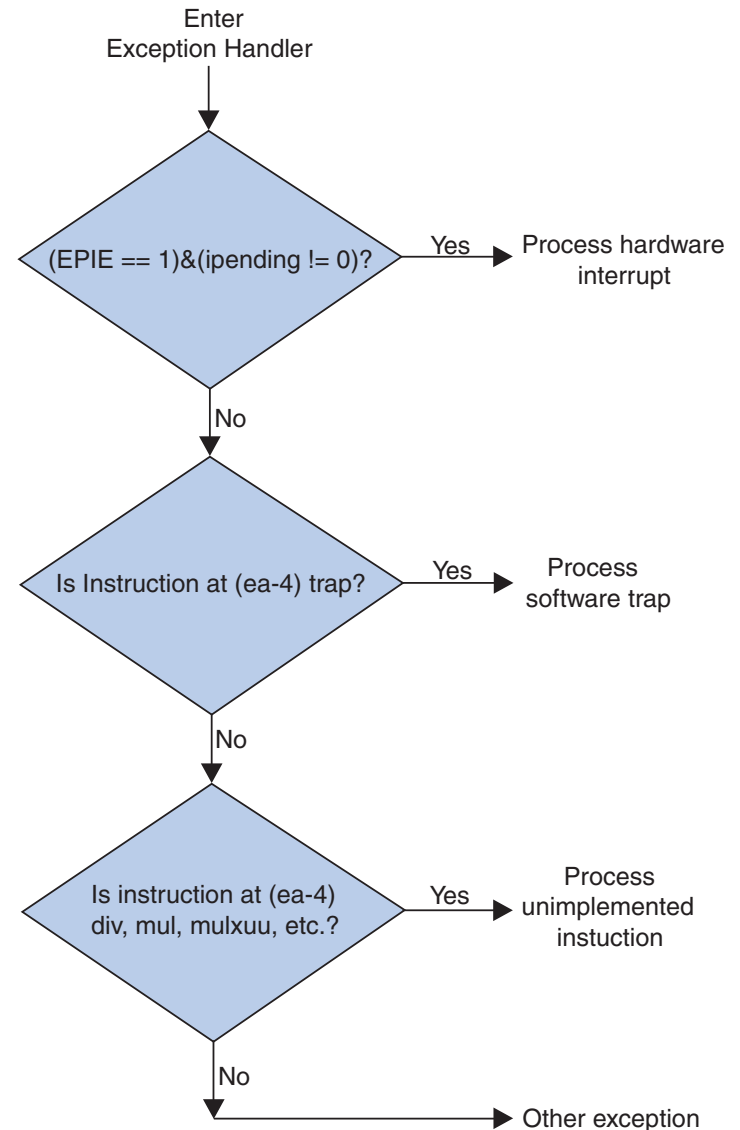
A slow peripheral device 'tells' the microprocessor when it is ready to receive or send new data.

For the Nios II processor Hardware Interrupts provide the mechanism for external events to catch the attention of the processor using the Exception processing capability.

The Exception process

What happens before this?

The contents of the Program Counter must be saved (goes to ea)
together with the contents of the Status Register (goes to et).





Code for Dealing with an Exception

EXCEPTION_HANDLER:

```
subi    sp, sp, 16          /* make room on the stack */
stw     et, 0(sp)           /* store the exception temporary register on stack*/
rdctl   et, ctl4            /* get pending interrupt bits into et*/
beq     et, r0, SKIP_EA_DEC  /* interrupt is not external if no bits set*/
subi    ea, ea, 4           /* dec exception return address for Trap or ui */
```

SKIP_EA_DEC:

```
stw     ea, 4(sp)           /* save all used registers on the Stack */
stw     ra, 8(sp)           /* needed if call inst is used */
stw     r22, 12(sp)         /* we will be using r22*/
rdctl   et, ctl4            /* get pending interrupt bits into et again!*/
bne     et, r0, CHECK_LEVEL_0 /* interrupt is an external interrupt */
```

NOT_EI: /* exception must be unimplemented instruction or TRAP */

```
br      END_ISR            /* instruction. This code does not handle those cases */
```

CHECK_LEVEL_0:

```
andi    r22, et, 0b1       /* interval timer is interrupt level 0 */
beq     r22, r0, CHECK_LEVEL_1 /* check bit 0 of pending interrupt bits*/
call    INTERVAL_TIMER_ISR /* if not set then skip*/
br      END_ISR            /* if set then call service routine*/
/* on return tidy up*/
```

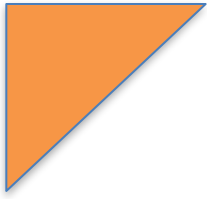


Code for Dealing with an Exception

```
CHECK_LEVEL_1:                /* pushbutton port is interrupt level 1 */
    andi    r22, et, 0b10      /* check bit 1 of pending interrupt bits */
    beq     r22, r0, END_ISR    /* other interrupt levels are not handled in this code */
    call    PUSHBUTTON_ISR

END_ISR:
    ldw     et, 0(sp)           /* restore all used register to previous values */
    ldw     ea, 4(sp)
    ldw     ra, 8(sp)           /* needed if call inst is used */
    ldw     r22, 12(sp)
    addi    sp, sp, 16          /* release stack space */

    eret                        /* return from exception code */
.end
```



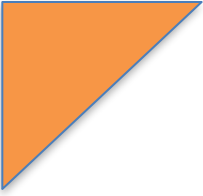
Evaluation of Interrupt I/O

Advantages

- Normal code program can be written with no regard for I/O operations.
- If no I/O operations occur then no CPU time is used to support I/O.
- Multiple levels of interrupt can be organised so that urgent I/O tasks can interrupt lower priority tasks.

Disadvantages

- System hardware is more complex
- There is a fixed time overhead for context switching which limits maximum throughput.



L03-4

Direct Memory Access

Clive Maynard



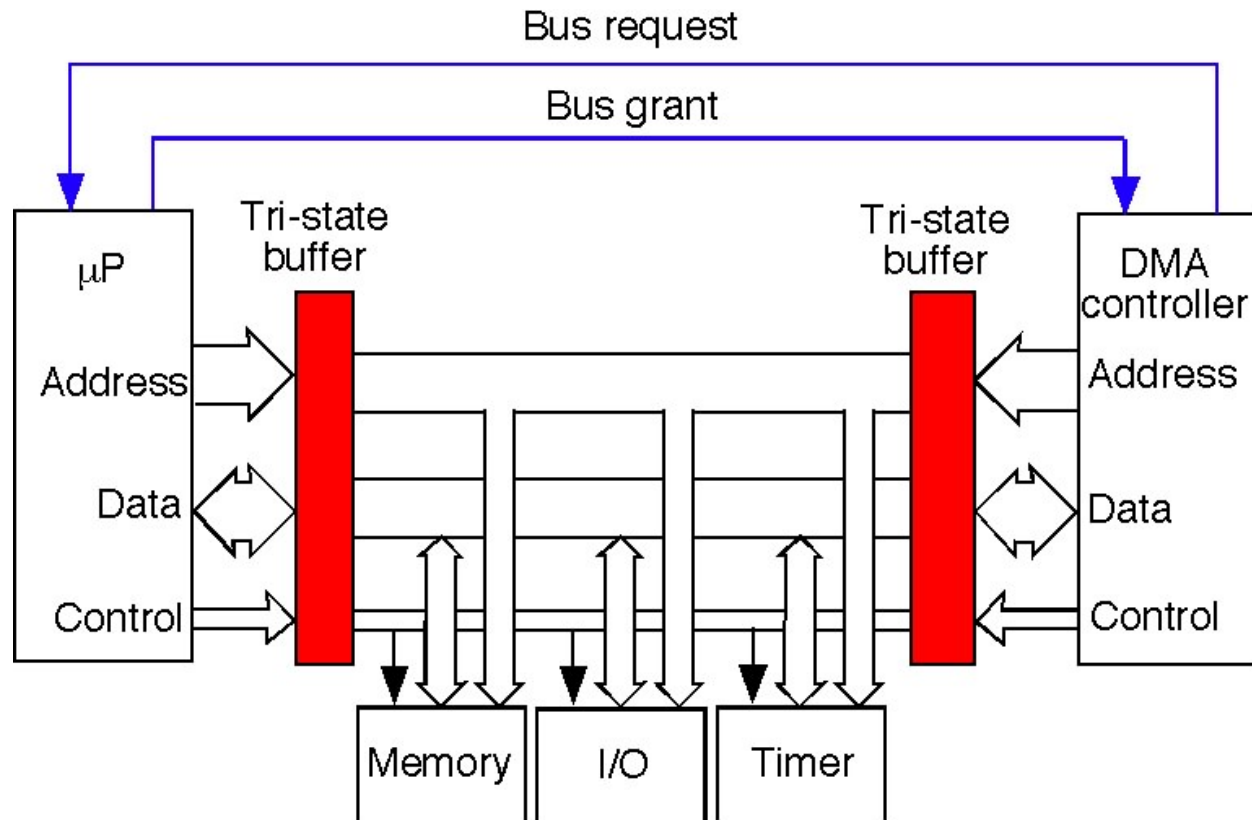
Direct Memory Access (DMA)

Extra hardware (a DMA controller) monitors service requests from peripheral devices and takes over the data transfer function of the microcomputer.

DMA can transfer data between a peripheral device and memory or between memory and memory.

Hardwired logic can perform a simple sequence of memory and/or I/O operations much faster than a microcomputer obeying a sequence of program instructions to do the same sequence.

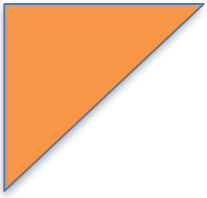
DMA Controller





Processor Halt DMA

- 1) Peripheral device signals the DMA controller that it needs service,
- 2) DMA controller shuts down the computer and takes over the bus (using HOLD and HLDA lines in the case of the 8086). HOLD or Bus Request is a signal from the DMA controller to the computer requesting control of the bus. Hold Acknowledge (HLDA) or Bus Grant tells the DMA controller that it has control of the bus.
- 3) DMA controller performs the data transfer to or from the peripheral (details of the transfer are programmed into the DMA controller or stored in memory), and
- 4) The computer is allowed to regain control of the bus by removing the HOLD signal.



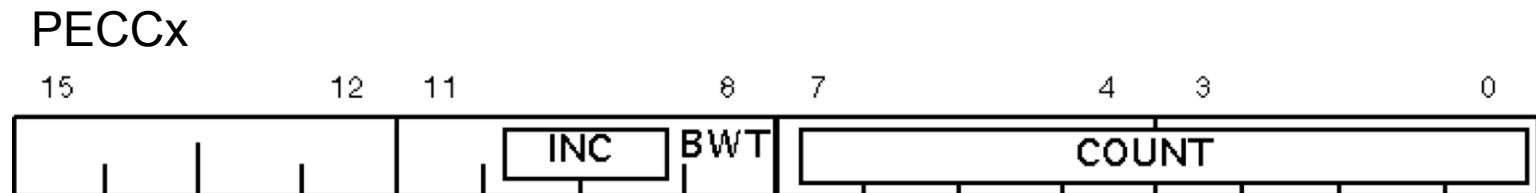
Interleaved DMA

If the computer has cached instructions and data it may be able to continue processing while the DMA system has control of the bus. If the processor really needs access to the bus, but the DMA controller is still using it then the processor will stall and must wait for access to the bus.

An alternative method is called interleaved DMA which only uses the system bus when the computer does not require it.

C167 Peripheral Event Controller

The peripheral event controller will do single byte/word transfers between two memory locations in the Infineon C167 in response to interrupts. For any of the 8 PEC channels two memory locations hold the address where data is to be read on interrupt (SRCPx) and the address where it is to be written (DSTPx). The PECCx register controls the action that is performed by the respective channel.



BWT (Byte/Word transfer selection) - 0 = word, 1 = byte

INC (Increment control) - 00 = pointer not modified, 01 = increment DSTPx by 1 or 2 (BXT), 10 = increment SRCPx by 1 or 2 (BXT), 11 = Reserved.



COUNT - PEC Transfer Count

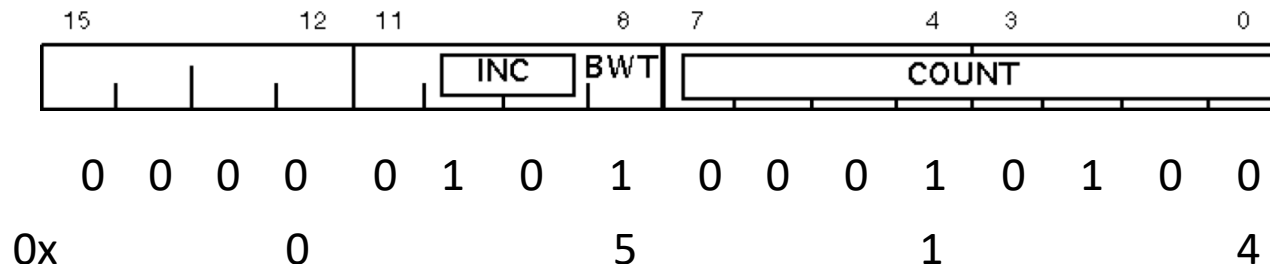
Previous COUNT	Modified COUNT	IR after PEC service	Action of PEC channel
0xFF	0xFF	0	Move a byte/word Continuous transfer mode (COUNT not modified)
<u>0xFE .. 0x02</u>	<u>0xFD .. 0x01</u>	0	Move a byte/word and decrement COUNT
0x01	0x00	1	Move a byte/word. Leave request flag set which triggers another request
0x00	0x00	1	No action. Activates service routine rather than PEC channel.

Setting Up the PEC

For example setting up PEC0 to transfer 20 bytes of data from the serial interface:

```
mov r0,#S0RBUF      ;data will come from receive buffer
mov SRCP0,r0
mov r0,#01000h      ;data sent to RAM memory starting at 01000h
mov DSTP0,r0
mov r0,#0514h       ;inc pointer, transfer byte, transfer 20 bytes
mov PECC0,r0
```

The basic response time for the PEC is 2 instruction cycles. In the best case with minimum sized instructions executing from internal ROM this equates to 3 T-states (150ns @20MHz clock).





Next

Next we will look at some of the practicalities of how information is moved around inside a computer and how it is stored in memory devices.

In practice most computers uses busses (parallel groups of wires) to transfer control signals, memory addresses and data.

This information will help you to interface additional memory and input/output devices to a microcontroller system.